# 卒 業 研 究 報 告

題　　目

コンピュータによる乱数発生と

ランダム・ウオーク

---


指 導 教 員

山本　哲也教授

報 告 者

楠葉　英樹


平成 14年　 2月　 5日


高知工科大学 電子・光システム工学科

目次

# １．　はじめに

本稿では、Ｃ言語による基本的な算法プログラムを利用し、乱数発生法について検討した。乱数は確率現象をシミュレーションするために欠かせないものである。また、シミュレーションの結果は乱数の精度に大きく影響される。そこで、得られた乱数が良い乱数か悪い乱数かを検定することに重点を置いた。

## １．１　目的

乱数発生法に、算術乱数がある。今回は、その中の合同法に着目し、混合合同法と乗算合同法の2つを用いて乱数発生を行った。

合同法の性質を知ると共に、混合合同法と乗算合同法のどちらの方が乱数発生法として優れているのかを、複数の検定法により明らかにすることを目的とした。

また、最後に最も良いと考えるものでシミュレーションを行った。

## ２．乱数の発生

本章では、これまでに良く使われてきた乱数発生法について議論する。また、その発生方法により発生した乱数が、乱数として認められるかの検定を行う。

### ２．１　乱数とは

乱数とは、サイコロを振って得られるような、無秩序に現れる数（現れる数の間に何の関係も無い）の事を言う。しかしながら、乱数を多数必要とする場合、サイコロを振って乱数を得るのは大変な労力や時間等を必要とし、現実的ではない。そこで、算術乱数をコンピュータに計算させることにより乱数を作り出す方法が考えられた。しかし、算術（計算）を用いて乱数を作り出す以上、現れる乱数同士が、全く無秩序というわけではない。よって、算術を用いて得られた乱数を、擬似乱数と呼ぶ。

擬似乱数の満たすべき条件として以下のことが挙げられる。

(a)　乱数発生法は多数個の乱数を速やかに発生できるものであること。

(b)　発生する乱数に周期があるとき、その周期は十分に長いものでなくてはならない。

(c)　再現性がある。初期条件が分かっていれば、いつ何時乱数を発生させても同じ結果が得られる。

(d)　発生した乱数は、良好な統計的性質を持たなくてはならない。

上記のうち、算術乱数において(a)はコンピュータを用いることにより、(c)は算術乱数が計算によって得られる以上、やり方と初期条件が分かっていればいつ何時やっても同じ結果が得られる。また、(b)・(d)においては、適宜検定を行うことが必要である。

また、本稿では擬似乱数も乱数と呼ぶこととする。

### ２．２　ランダム･ウォーク

ランダム･ウォークのシミュレーションは、酔っ払いが千鳥足で歩いているようなモデルを考える。つまり、一歩ごとに方向が定まらず、どの歩も独立している。このようなシミュレーションは、ブラウン運動（Brownian motion）や、金属中の電子の輸送現象のような、粒子のランダムな動きなど物理現象のシミュレーションに用いられる。

### ２．３　合同法乱数

一様乱数の生成方法として最も広く使われてきたのは、レーマー（Lehmer）が 1948 年頃に発表した線形合同法（linear congruential method）であろう。これは、漸化式

$$x_n = a \cdot x_{n-1} + b \quad (\text{mod M}) \qquad (2.3.1)$$

で得られる非負整数列$x_n$を、Mで割ることにより区間$[0,1)$の乱数を、$M/10$で割ることにより区間$[0,9]$の乱数を得る。$b=0$の場合を乗算合同法、b≠0 の場合を混合合同法と呼ぶ。a は乗数、ｂは加数、M は係数を示し、n=1,2,3,･･･である。また、n=1 のとき右辺の$x_{n-1}$は$x_0$であり、この$x_0$は初期値である。

## ２．４　乗算合同法

乗算合同法とは、２数の積をある数で割った余りをとる方法である。漸化式

$$x_n = a \cdot x_{n-1} \quad (\text{mod M}) \qquad (2.4.1)$$

で表される。この漸化式により、数列$x_1, x_2, x_3$･･･を発生させる。a は乗数、M は係数を示す（a、M 共に整数）。なお、M の選択方法については後で述べることにする。

ここで、乗算合同法の計算例を示す。

例.　乗算合同法を用いて、一桁の乱数を作る。ただし、乗数 a=3045、係数 M=34768、初期値$x_0$=257 の値を用いる。

【計算例】

漸化式　$x_n = a \cdot x_{n-1}$　(mod M)　において、n=1 から$x_1, x_2, x_3$･･･を計算する。実際には、

$$\begin{aligned} x_1 &= a \cdot x_0 \quad (\text{mod M}) \\ &= 3045 \times 257 \quad (\text{mod } 34768) \\ &= 3045 \times 257 - \left[ \frac{3045 \times 257}{34768} \right] \times 34768 \\ &= 3045 \times 257 - 22 \times 34768 \\ &= 17669 \end{aligned} \qquad (2.4.2)$$

となる。[　]はガウス記号を表し、[　]内の数値の整数部を表している。

$$\frac{3045\times257}{34768} \fallingdotseq 22.5082 \tag{2.4.3}$$

なので、

$$\left[\frac{3045\times257}{34768}\right] = 22 \tag{2.4.4}$$

となる。

また、

$$\begin{aligned} x_2 &= a\cdot x_1 \pmod{\mathrm{M}} \\ &= 3045\times17669 \pmod{34768} \\ &= 3045\times17669 - \left[\frac{3045\times17669}{34768}\right]\times34768 \\ &= 3045\times17669 - 1547\times34768 \\ &= 16009 \end{aligned} \tag{2.4.5}$$

このようにして数列 $x_1, x_2, x_3\cdots$ を求める。区間 $[0,9]$ の乱数 $y_n$ は、$x_n$ を M を 10 で割った値で割り、得られた値の整数部をもちいることによって得られる。

$$y_n = \left[\frac{x_n}{M/10}\right] \tag{2.4.6}$$

実際には、

$$\begin{aligned} y_1 &= \left[\frac{x_1}{34768/10}\right] \\ &= \left[\frac{17669}{3476.8}\right] \\ &= 5 \end{aligned} \tag{2.4.7}$$

$$y_2 = \left\lfloor \frac{x_2}{34768 \diagup 10} \right\rfloor$$
$$= \left\lfloor \frac{16009}{3476.8} \right\rfloor$$
$$= 4 \qquad (2.4.8)$$

このように、順次区間$[0,9]$の乱数$y_n$を求めていく。

区間$[0,1)$の乱数$z_n$は、$x_n$をMで割ることにより得られる。

$$z_n = \frac{x_n}{M} \qquad (2.4.9)$$

実際には、

$$z_1 = \frac{x_1}{34768}$$
$$= \frac{17669}{34768}$$
$$\fallingdotseq 0.508197 \qquad (2.4.10)$$

$$z_2 = \frac{x_2}{34768}$$
$$= \frac{16009}{34768}$$
$$\fallingdotseq 0.460452 \qquad (2.4.11)$$

このように、順次区間$[0,1)$の乱数$z_n$を求めていく。

## ２．５　混合合同法

混合合同法とは、２数の積にある数を加えたものをある数で割った余りをとる方法である。漸化式

$$x_n = a \cdot x_{n-1} + b \quad (\text{mod M}) \tag{2.5.1}$$

で表される。この漸化式により、数列$x_1, x_2, x_3$・・・を発生させる。a は乗数、b は加数、M は係数を示す（a、b、M 共に正整数)。なお、M の選択方法については後で述べることにする。

ここで、混合合同法の計算例を示す。

例. 混合合同法を用いて、一桁の乱数を作る。ただし、乗数 a=3045、加数 b=30120,
係数 M=34768、初期値$x_0$=257 の値を用いる。

【計算例】
漸化式 $x_n = a \cdot x_{n-1} + b \quad (\text{mod M})$ において、n=1 から$x_1, x_2, x_3$・・・を計算する。

実際には、

$$\begin{aligned} x_1 &= a \cdot x_0 + b \quad (\text{mod M}) \\ &= 3045 \times 257 + 30120 \quad (\text{mod } 34768) \\ &= \left(3045 \times 257 + 30120\right) - \left[\frac{3045 \times 257 + 30120}{34768}\right] \times 34768 \\ &= \left(3045 \times 257 + 30120\right) - 23 \times 34768 \\ &= 13021 \end{aligned} \tag{2.5.2}$$

となる。[ ]はガウス記号を表し、[ ]内の数値の整数部を表している。

$$\frac{3045 \times 257 + 30120}{34768} \fallingdotseq 23.3745 \tag{2.5.3}$$

なので、

$$\left[\frac{3045 \times 257 + 30120}{34768}\right] = 23 \tag{2.5.4}$$

となる。
また、

$$
\begin{aligned}
&x_2 = a \cdot x_1 + 30120 \quad (\text{mod M}) \\
&= 3045 \times 13021 + 30120 \quad (\text{mod } 34768) \\
&= \left(3045 \times 13021 + 30120\right) - \left[\frac{3045 \times 13021 + 30120}{34768}\right] \times 34768 \\
&= \left(3045 \times 13021 + 30120\right) - 1141 \times 34768 \\
&= 8777
\end{aligned}
\tag{2.5.5}
$$

このようにして数列 $x_1, x_2, x_3 \cdots$ を求める。区間[0,9]の乱数 $y_n$ は、乗算合同法の場合と同じ方法で求められる。つまり、

$$
\begin{aligned}
y_1 &= \left[\frac{13021}{3476.8}\right] \\
&= 3
\end{aligned}
\tag{2.5.6}
$$

$$
\begin{aligned}
y_2 &= \left[\frac{8777}{3476.8}\right] \\
&= 2
\end{aligned}
\tag{2.5.7}
$$

このように、順次一桁の乱数 $y_n$ を求めていく。この乱数 $y_n$ は、区間[0,9]の乱数となる。

区間 $[0,1)$ の乱数 $z_n$ も、乗算合同法の場合と同じようにして得られる。

$$
z_n = \frac{x_n}{M} \tag{2.5.8}
$$

実際には、

$$
\begin{aligned}
z_1 &= \frac{x_1}{34768} \\
&= \frac{13021}{34768}
\end{aligned}
$$

$\fallingdotseq 0.374511$

(2.5.9)

$$z_2 = \frac{x_2}{34768} = \frac{8777}{34768} \fallingdotseq 0.252445 \qquad (2.5.10)$$

このように、順次区間$[0,1)$の乱数$z_n$を求めていく。

## ２．６　法 M の選択

線形合同法　$X_n = aX_{n-1} + b \pmod{M}$　から生成される非負整数列$X_n$の周期は、M より長くはなりえない。したがって、M はかなり大きくとっておく必要がある。そして、疑似乱数において、周期は M に等しくなることが望ましい。

よく行われる選択法としては、使用する計算機やプログラム言語で表現可能な最大の正整数をとる。実際には、使用するコンピュータやプログラム言語の能力による。今回は、C 言語を使用するので、取り得る最大の正整数である　$M = 2^{31} - 1 = 2147483647$ を採用する。

また、M の変化が乱数に与える影響を考察するために、$M=32768(=2^{15})$、$256(=2^8)$　を用いて乗算合同法と混合合同法の比較を行いました。

## ２．７　合同法の比較

### ２．７．１　一様性の検定

一様とはどの数値においても出現頻度の等しいことを言う。一様乱数とは、ある区間においてどの数値も同じ確立で出現する乱数である。たとえば、さいころの目は 1 から 6 まであり、サイコロを振ったときに、どの目も$1/6$の確立で出現する。

乱数が一様性を持っているとき、

$$\text{平均} \qquad \sum_{n-1}^{N} \frac{x_n}{N} = \frac{1}{2} \qquad (2.7.1)$$

2 次積率　$\sum_{n-1}^{N} \frac{x_n^2}{N} = \frac{1}{3}$　(2.7.2)

3 次積率　$\sum_{n-1}^{N} \frac{x_n^3}{N} = \frac{1}{4}$　(2.7.3)

の値に近づくことが知られている。$x_n$ は区間$[0,1)$の乱数を、

N は乱数発生の試行回数$(n=1,2,3,\cdots,N)$を表している。以下に　、a=1229,b=9378，$x_0=257$，M=32768 のときの混合合同法と乗算合同法それぞれの一様性の検定を行い、グラフ化したものを示しました。

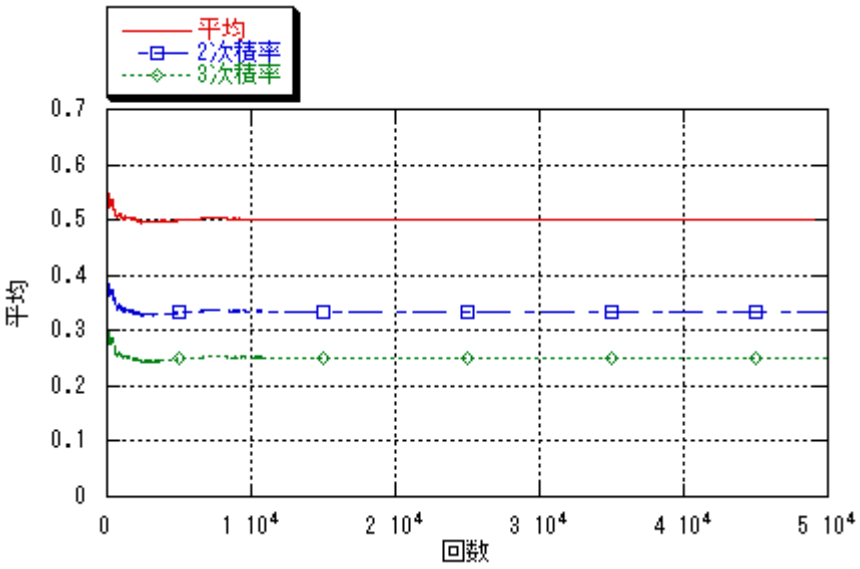


図 2.7.1.1　一様性の検定　　混合合同法

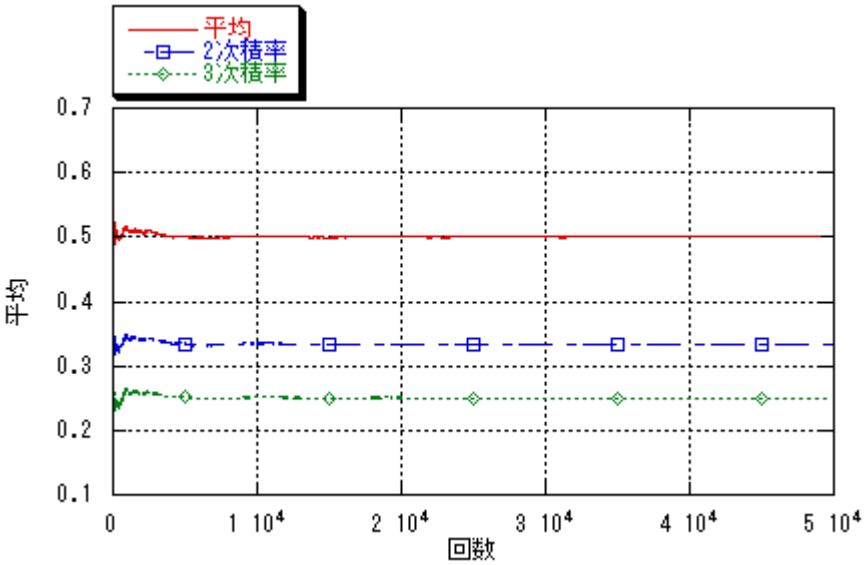


図 2.7.1.2　乱数の一様性の検定　乗算合同法

図 2.7.1.1・2.7.1.2 を見ると、どちらも試行回数 4000 回程で目的の値に収束しており、どちらも一様乱数と言える。

以下には、a=1229,b=9378，$x_0 = 257$，M=32768 のときの混合合同法と乗算合同法それぞれの一様性の検定を行い、グラフ化したものを示しました。

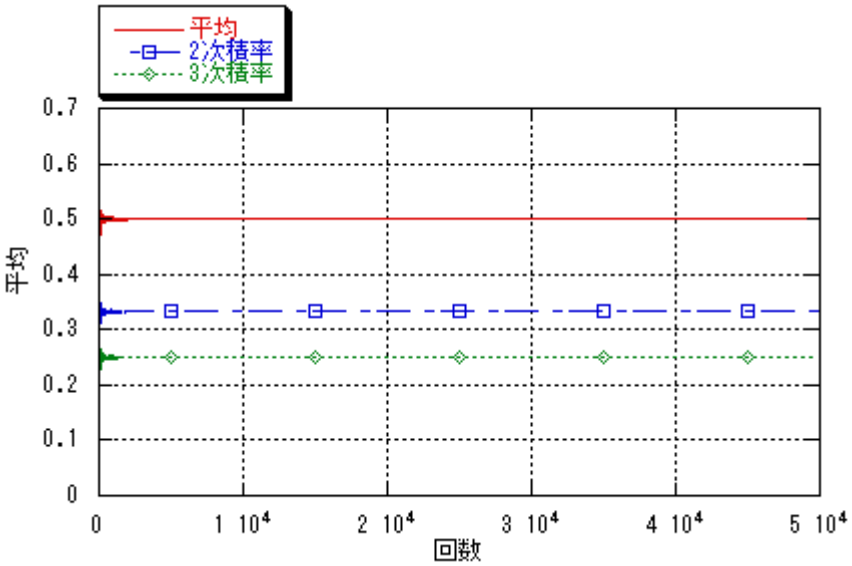


図 2.7.1.3　乱数の一様性の検定　混合合同法

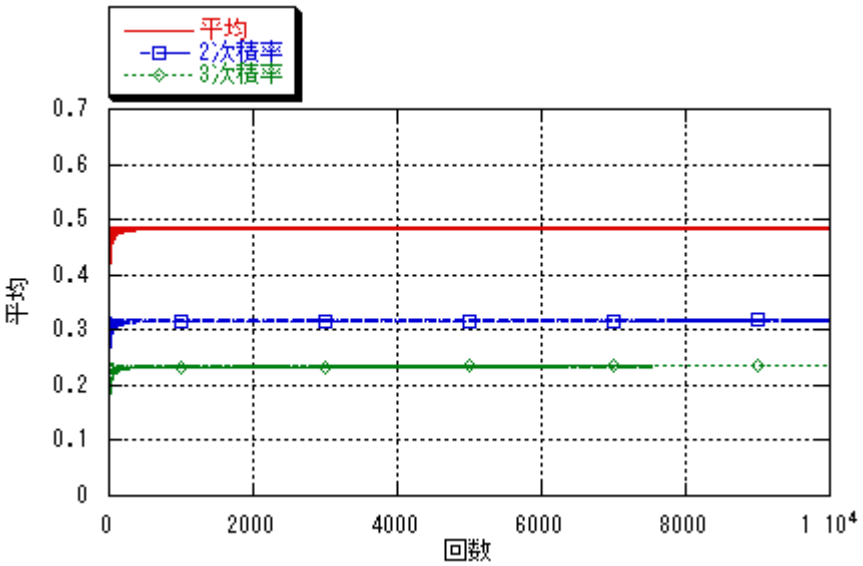


図 2.7.1.4　乱数の一様性の検定　乗算合同法

図 2.7.1.3･2.7.1.4 を見ると、試行回数 2000 回程で収束している。これは図３・４の場合が、M＝256 で先に述べたように、周期はM以下になるので、M以下の値で収束する。よって、図 2.7.1.3･2.7.1.4 場合が、とても早く収束しているのは、当然のことである。また、図 2.7.1.4 は目標値のやや下で収束している。よって、M が小さいときの乗算合同法において一様性が認められない場合がある。

## ２．７．２　乱数の周期

ここでは、得られた区間[0,9]の乱数 $y_n$ を用い、数直線上を

$y_n$ が奇数のとき右に 1

$y_n$ が偶数のとき左に 1

移動するというシミュレーションを行った。また、a=1229、b=9378、$x_0 = 257$ を使用し、M=32768($=2^{15}$)と M=256($=2^8$)を用いた。

まず、M=32768 のときの混合合同法と乗算合同法のシミュレーション結果を示す。

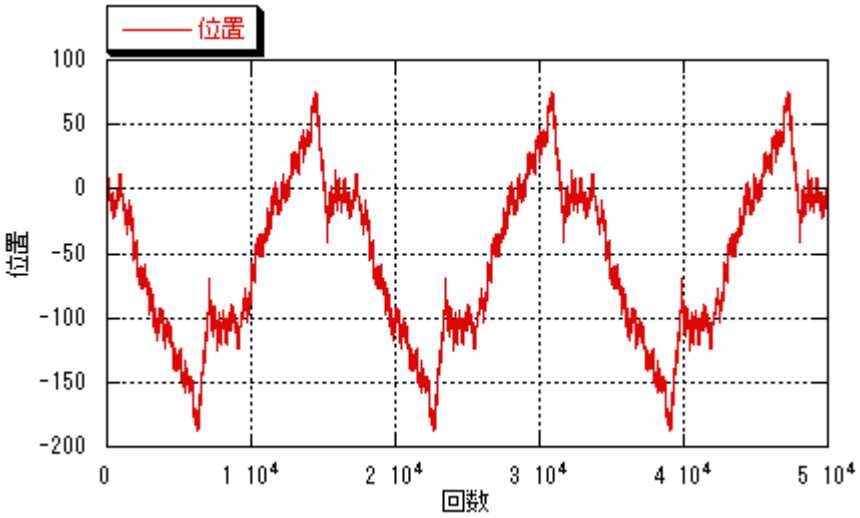


図 2.7.2.1　位置の推移　混合合同法

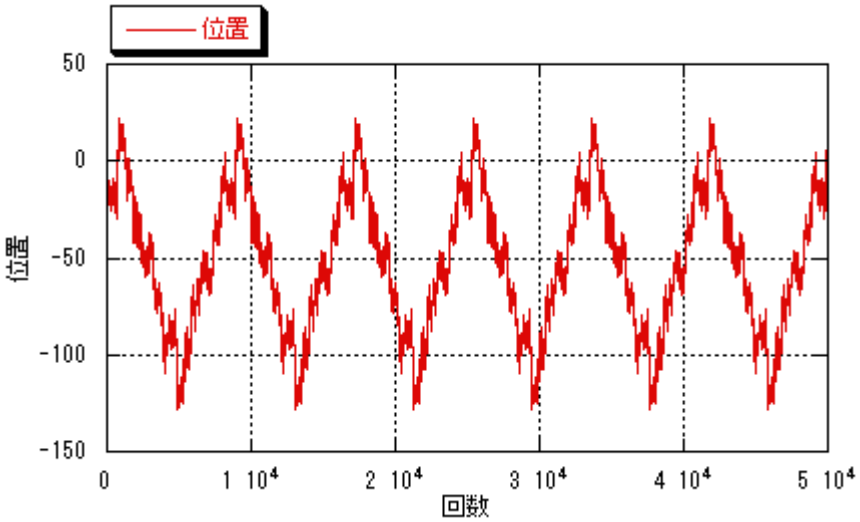


図 2.7.2.2　位置の推移　乗算合同法

次に、M=256 のときの混合合同法と乗算合同法のシミュレーション結果を示す。

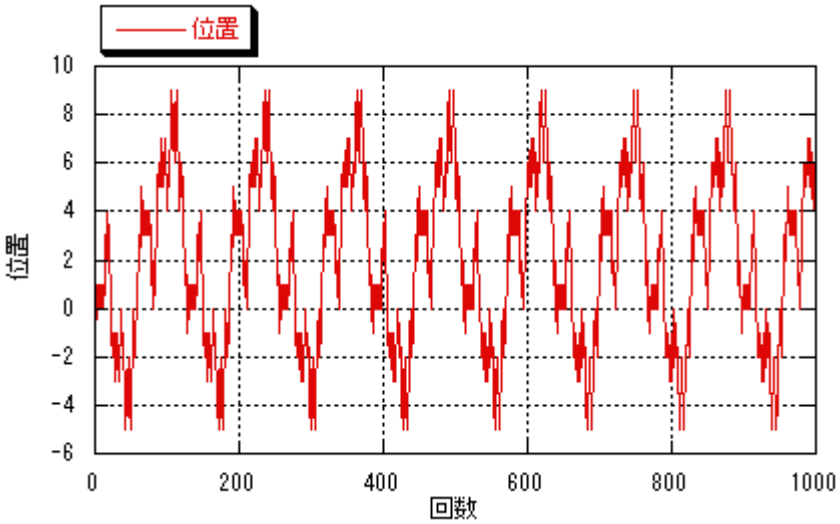


図 2.7.2.3　位置の推移　混合合同法

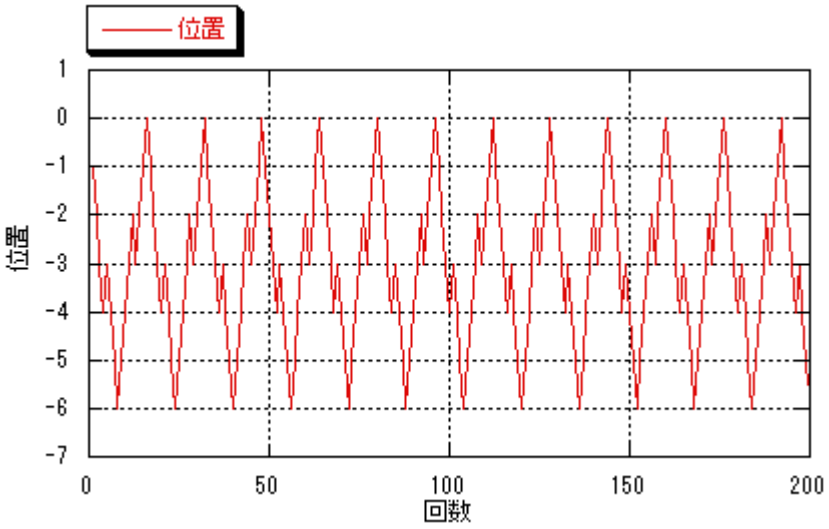


図 2.7.2.4　位置の推移　乗算合同法

図 2.7.2.1･2.7.2.2･2.7.2.3･2.7.2.4 を見てみると、同じ波形が繰り返されているのが見て取れる。これは、それぞれの乱数列の周期を示している。実際には、

図 5　　$16384\left(=2^{14}\right)$

図 6　　$8192\left(=2^{13}\right)$

図７　　$128\left(=2^{7}\right)$

図８では $16\left(=2^{4}\right)$

でそれぞれ一つの周期が終わっている。

この結果から、

１、周期は $2^{n}\left(n=0,1,2,\cdots\right)$ で表される

２、混合合同法の方が乗算合同法よりも周期が長い
また、
３、周期はM以下になることも確認できた
以上のことが解かった。

区間 $[0,1)$ の乱数を、縦軸を乱数、横軸を試行回数のグラフに点でプロットしてみると下図のようになる。

M=32768 $\left(=2^{15}\right)$ のとき、

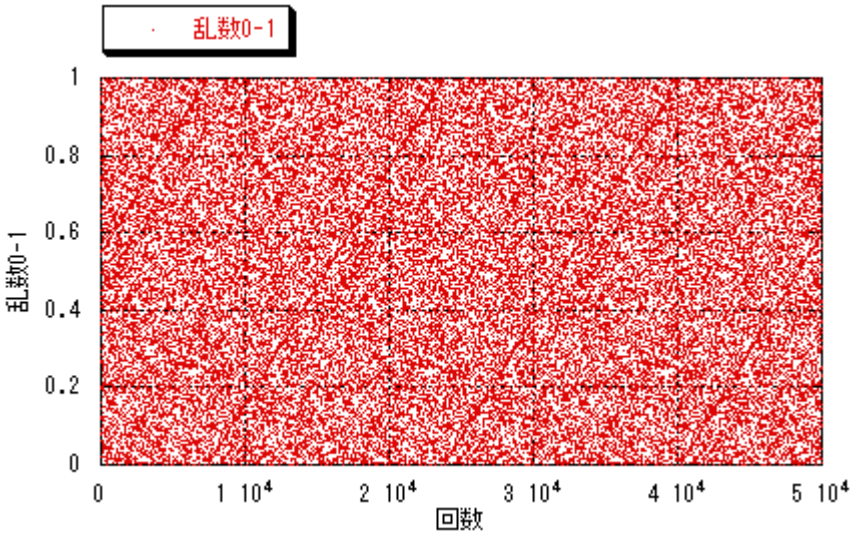


図 2.7.2.5　区間 $[0,1)$ の乱数　混合合同法

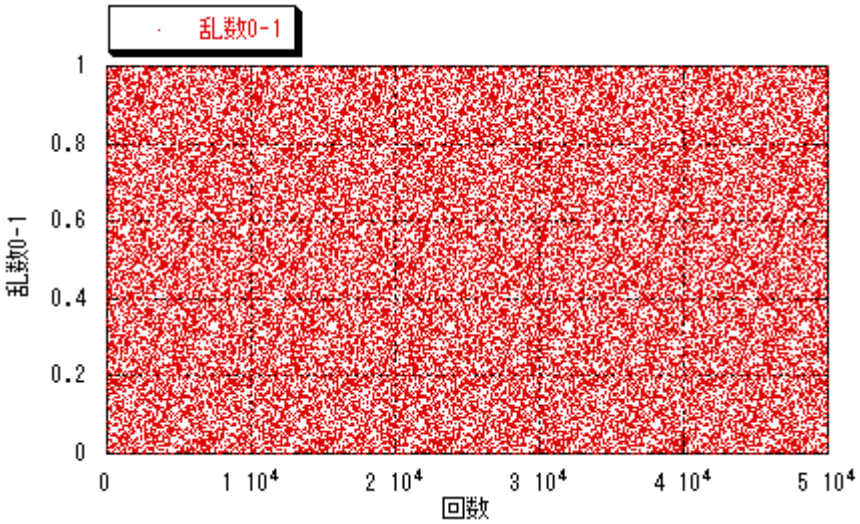


図 2.7.2.6　区間$[0,1)$の乱数　乗算合同法

次に、M=256$(=2^8)$の場合は以下のようになる。

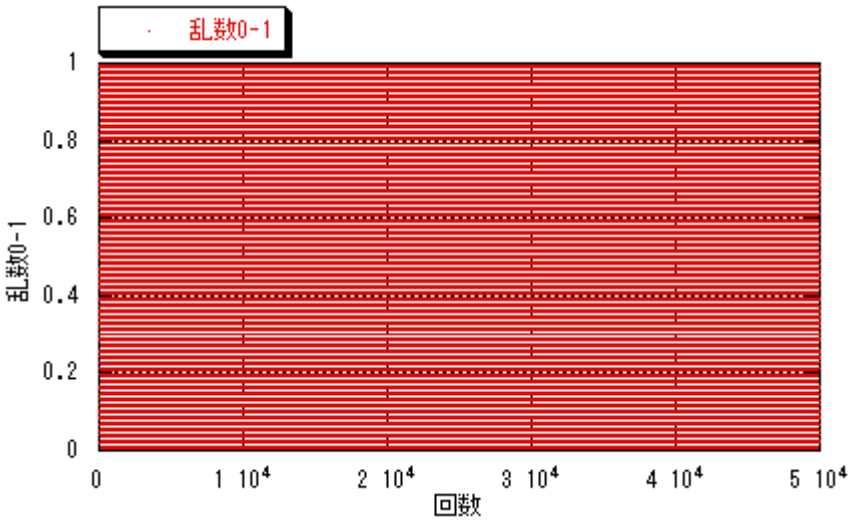


図 2.7.2.7　区間$[0,1)$の乱数　混合合同法

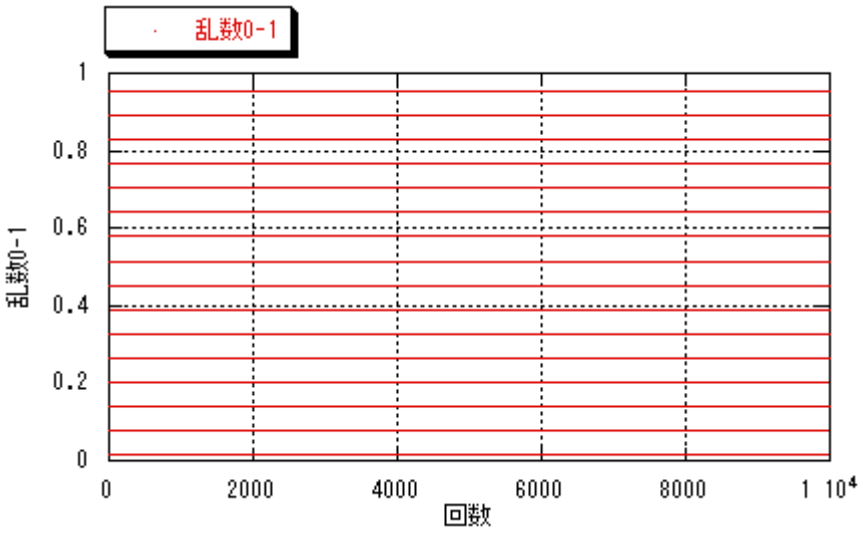


図 2.7.2.8　区間$[0,1)$の乱数　乗算合同法

図 2.7.2.5・2.7.2.6・2.7.2.7・2.7.2.8 の各グラフには線状の集合が見て取れる。これは、それぞれの乱数列の周期を表している。

同じ条件下において、乗算合同法と混合合同法は、一様性については同等、周期性については混合合同法の方が良いという結果に至った。よって、これ以降は、混合合同法についてのみ述べるものとする。

## 2．8　初期値の選択

初期値$x_0$が乱数にどのような影響があるのかを調べると共に、どのような数値が適しているのかを調べることにした。ここでは、

係数　$M=2147483674\left(=2^{31}-1\right)$

乗数　a=1229

加数　b=9378

と、おいた。また、初期値　$x_0$は、

256（$=2^8$）

5560（$\fallingdotseq 2^{12}$）

2743764（$\fallingdotseq 2^{21}$）

57405839（$\fallingdotseq 2^{26}$）

305673400（$\fallingdotseq 2^{28}$）

を任意に選び試してみた。

まず、初期値の変化によって周期が変化するかどうかを試す。

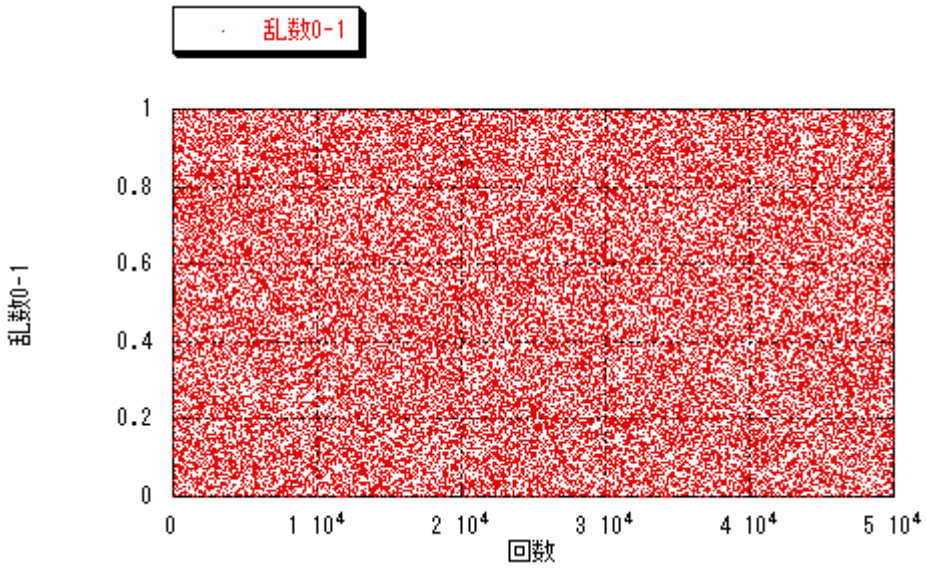


図 2.8.1　区間$[0,1)$の乱数　M=256

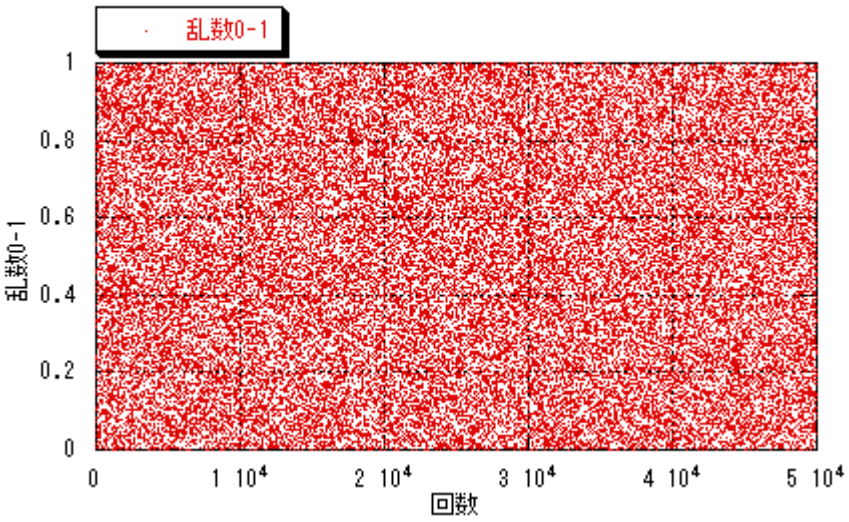


図 2.8.2　区間$[0,1)$の乱数　M=5560

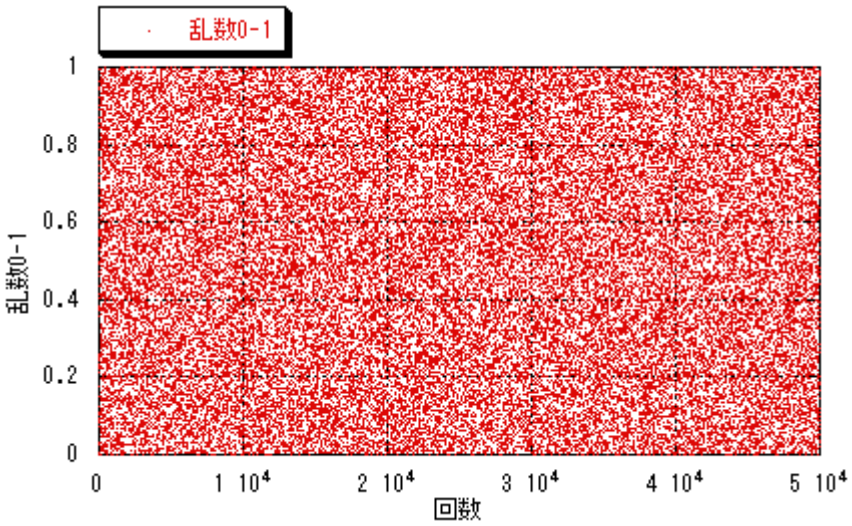


図 2.8.3　区間$[0,1)$の乱数　M=2743764

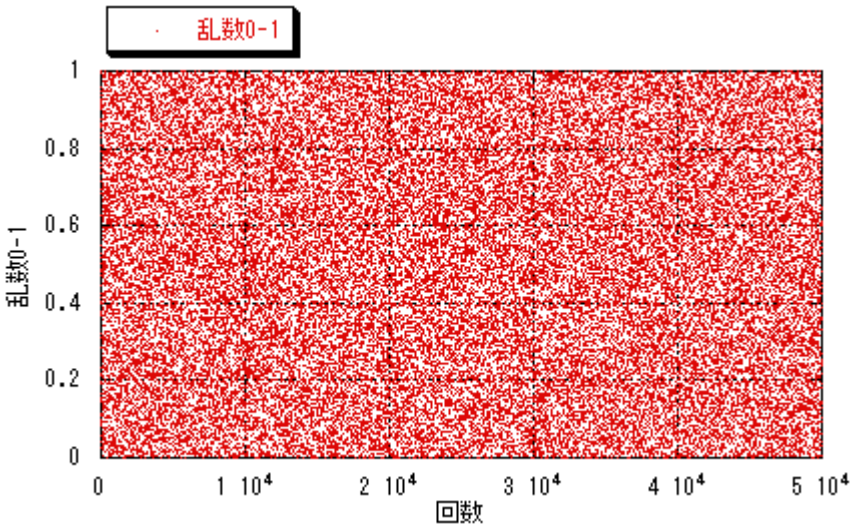


図 2.8.4　区間$[0,1)$の乱数　M=57405839

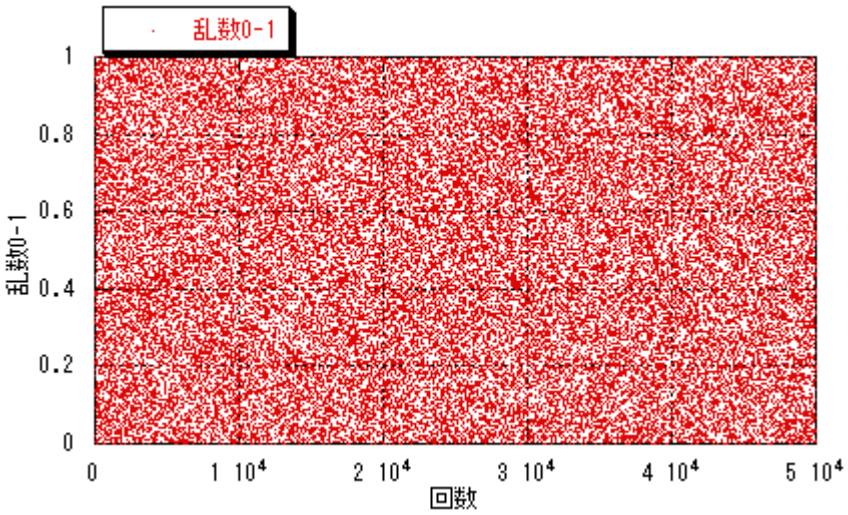


図 2.8.5　区間$[0,1)$の乱数　M=305673400

図 2.8.1･2.8.2･2.8.3･2.8.4･2.8.5 を見ると、乱数が特別疎なところや密なところが見られないことから、適度に長い周期を持っていると判断し得よう。また、初期値$x_0$によって極端に周期が短くなることが無いこともわかった。

次に、それぞれの場合においての一様性の検定を行った。

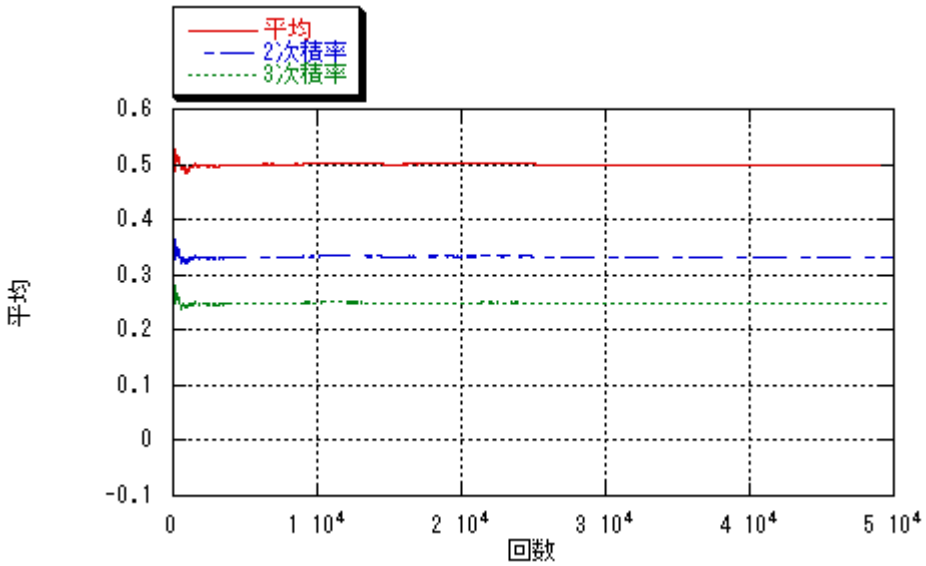


図 2.8.6　一様性の検定　$x_0$=256

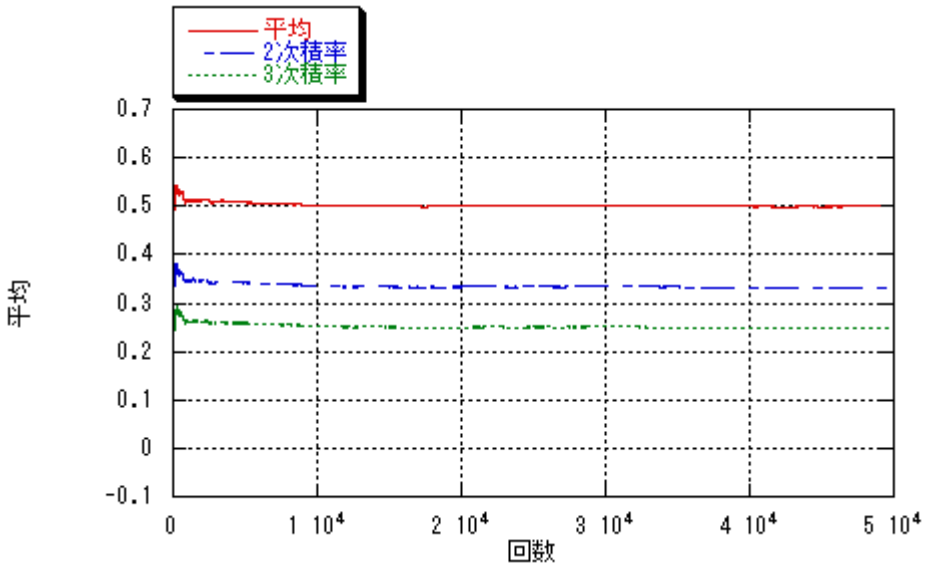


図 2.8.7　一様性の検定　$x_0$=5560

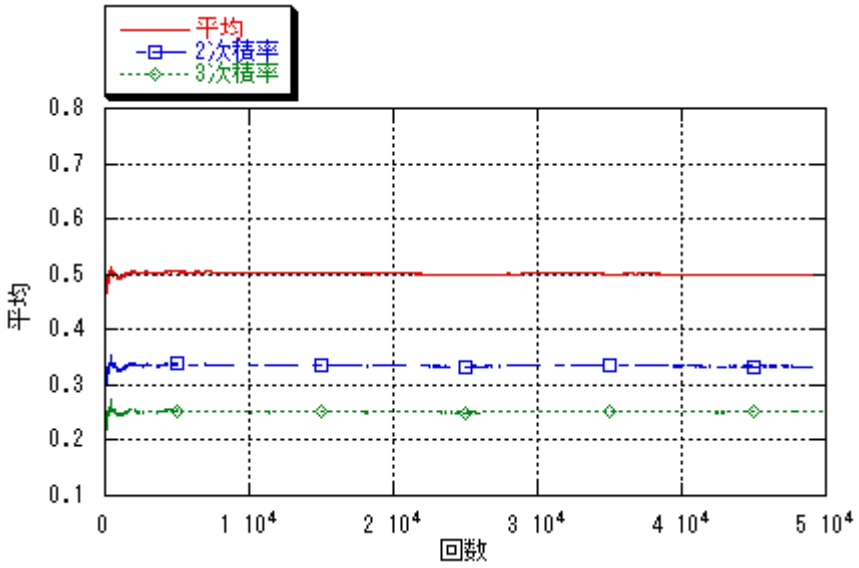


図 2.8.8　一様性の検定　$x_0$=2743764

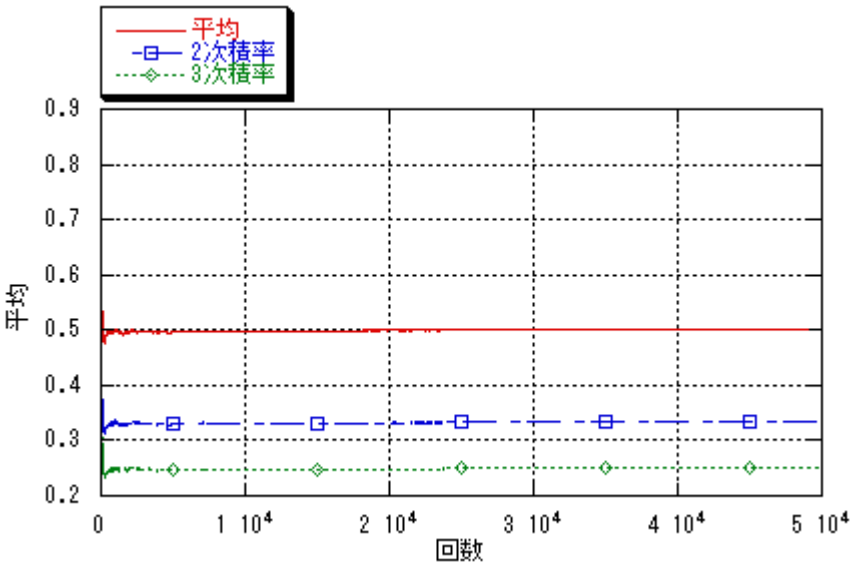


図 2.8.9　一様性の検定　$x_0$=57405839

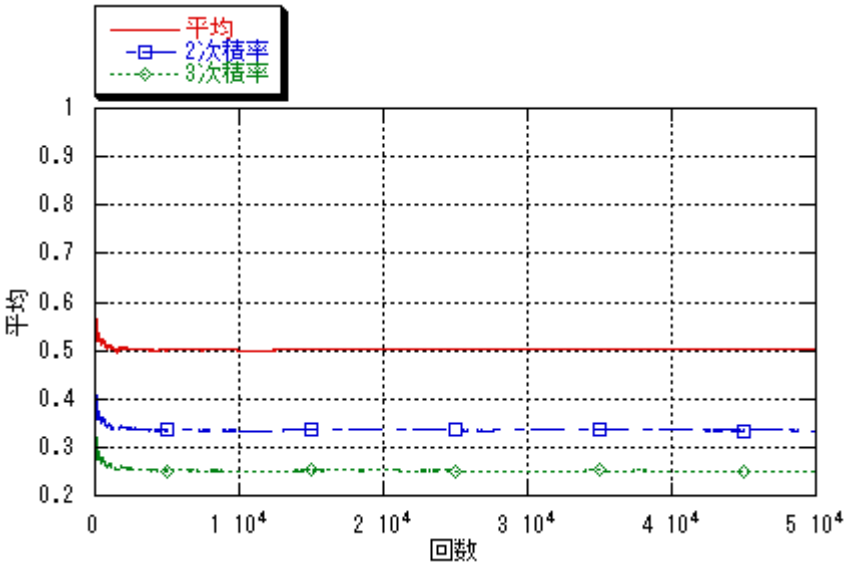


図 2.8.10　一様性の検定　$x_0$=305673400

図 2.8.6･2.8.7･2.8.8･2.8.9･2.8.10 を見てみると、どのグラフにおいても試行回数５000 回で収束がほぼ終わっている。この結果から初期値 $x_0$ の変化によって一様性が大きく乱されることはないことがわかった。

初期値を変えることにより、周期が極端に短くなったり、一様性が大きく乱れることはなかった。

次に、1 次元ランダム･ウォークを用いて、一様性乱数の検定を行いました。

乱数が理想的な一様性乱数であるならば、1 次元ランダム・ウォークの位置の平均は 0 に近くなる。下図はそれぞれの場合の位置の変化を表す。

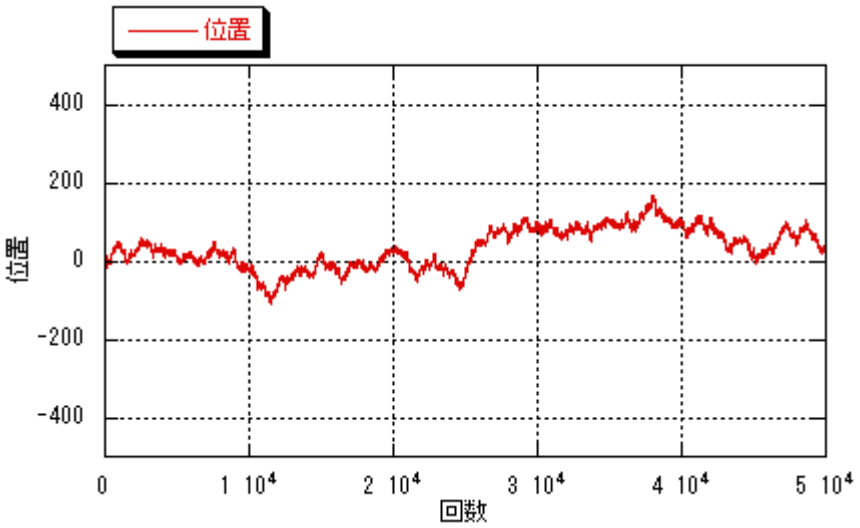


図 2.8.11　1 次元ランダム・ウォーク　$x_0$ =256 のとき、位置の平均は 33．285 になります。

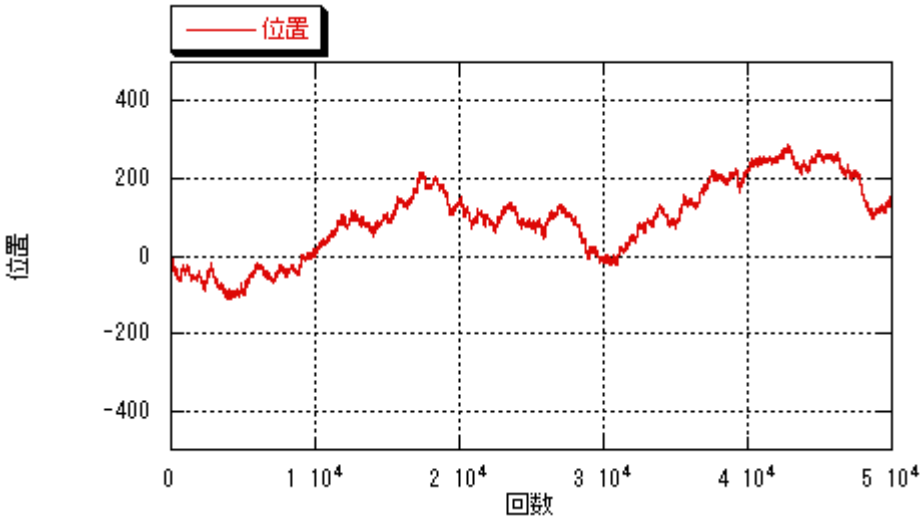


図 2.8.12　1 次元ランダム・ウォーク　$x_0$ =5560 のとき、位置の平均は 95．774 になります。

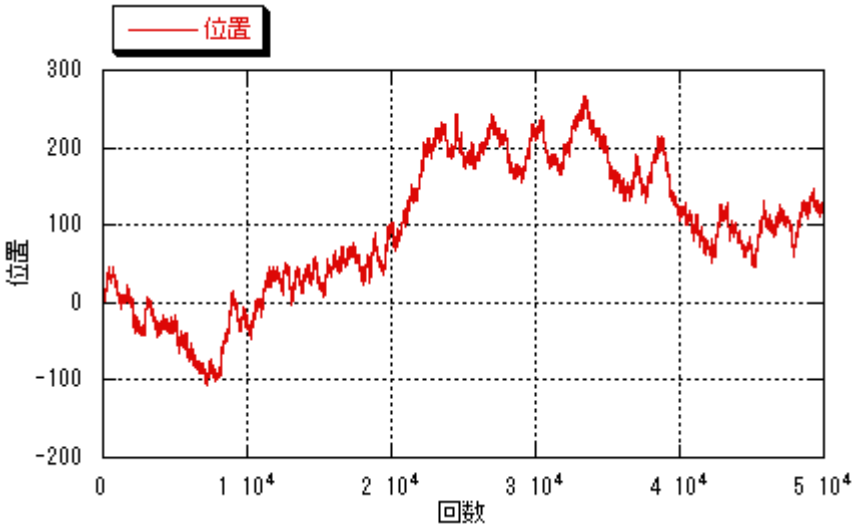


図 2.8.13　次元・ランダムウォーク　$x_0$=2743764

図 2.8.13　のとき、位置の平均は 94．484 になります。

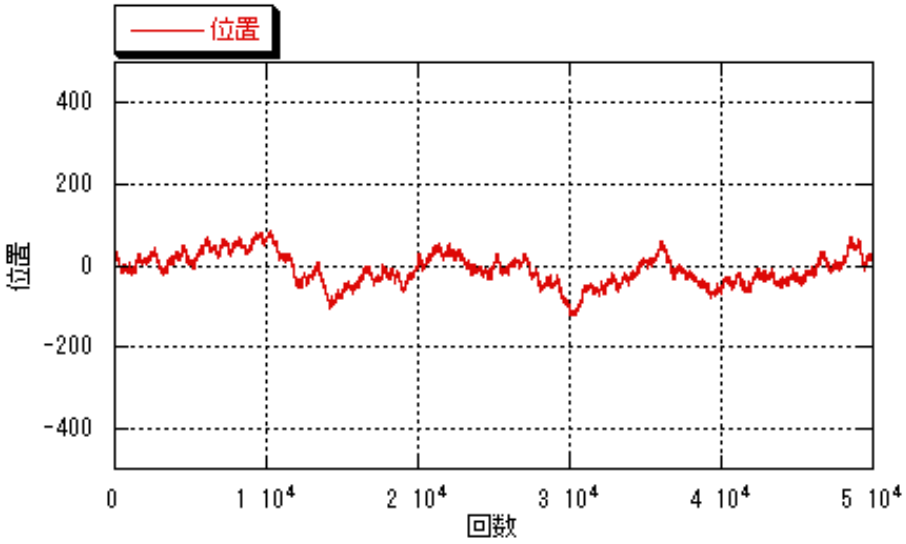


図 2.8.14　1 次元ランダム・ウォーク　$x_0$=57405839

図 2.8.14　のとき、位置の平均は-10．012 になります。

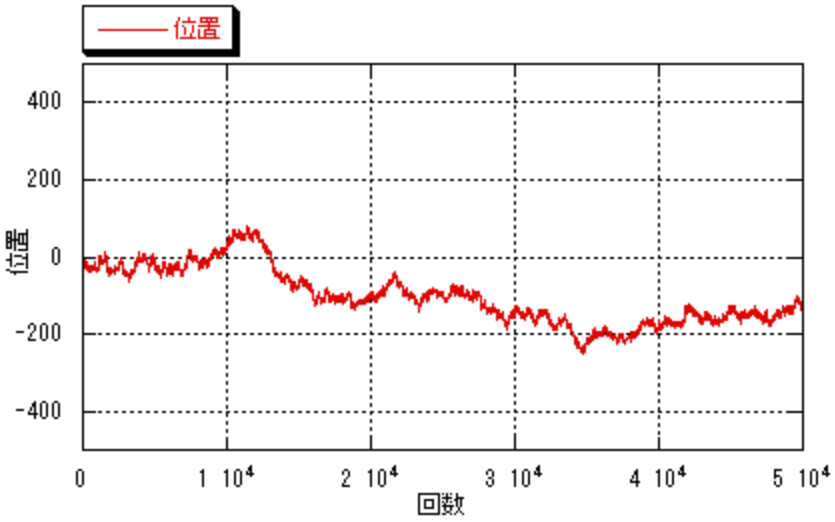


図 2.8.15　ランダム・ウォーク　$x_0$=305673400

図 2.8.15　のとき、位置の平均は-101．51 になります。

以上の結果から、図 2.8.14　の場合の位置の平均が最も 0 に近いので、

係数　M=2147483674$\left(=2^{31}-1\right)$

乗数　a=1229

加数　b=9378

の場合の初期値は、

初期値　$x_0=57405839$

が、適切であると考えた。

よって、本稿において、合同法乱数において最も良いのは、

混合合同法

係数　M=2147483674$\left(=2^{31}-1\right)$

乗数　a=1229

加数　b=9378

初期値　$x_0=57405839$

を用いることだと考えた。

## 3．　シミュレーション　2 次元ランダム・ウォーク

最後に、これまでの検定の結果などから私の最も良いと考える数値で 2 次元のランダム・ウォークのシミュレーションを行った。数値は、第 2 章で得られた、

M=2147483647
a=1229
b=9378
$x_0$ =57405839

を用いた。

ここでは、原点を出発し、$1/4$ の確立で上下左右に 1 ずつ移動する方法をとった。下図は、位置の移動を示している。

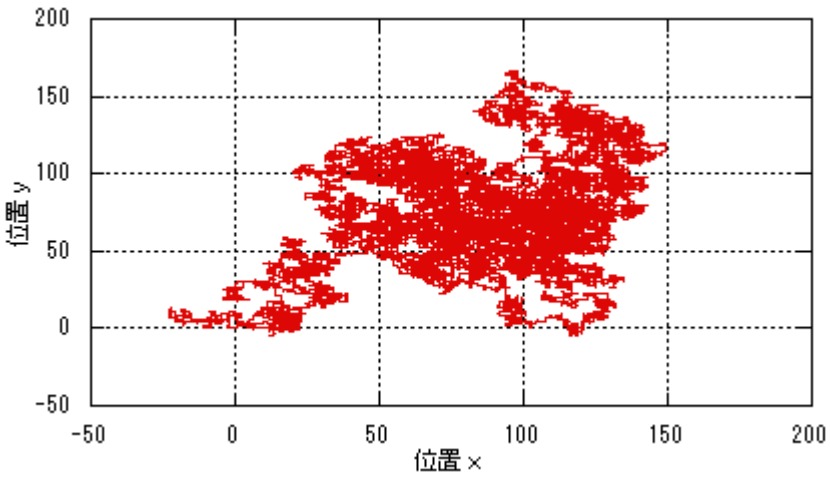


図 3.1　位置の変化

また、2 次元のランダム・ウォークの特性として、

$$R \fallingdotseq \sqrt{N}$$

（Rは原点からの距離、Nは試行回数）

が、知られている。

図 3.2　は試行回数 100000000（＝$10^8$）回の場合である。直線は、$R \fallingdotseq \sqrt{N}$

の理想的な直線であり、この直線に近いほど良い。

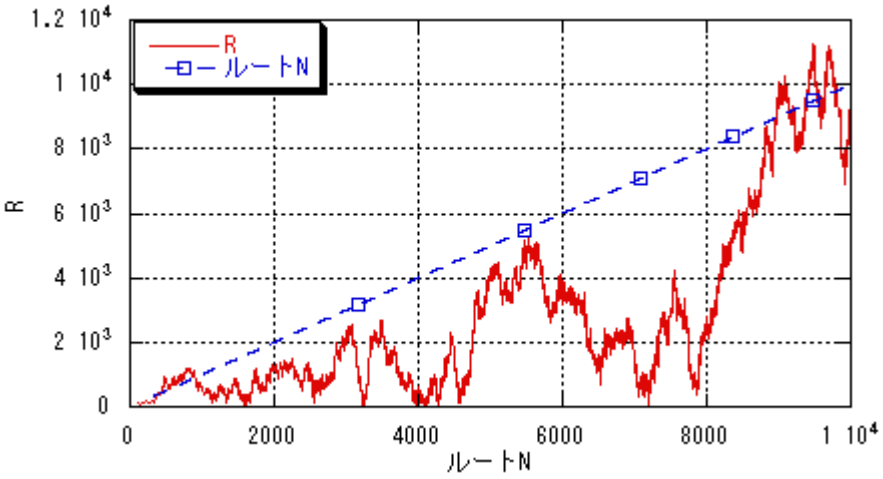


図 3.2　$R \fallingdotseq \sqrt{N}$ のグラフ　N＝$10^8$

下図は試行回数 1600000000（＝$16\times10^8$）回の場合である。

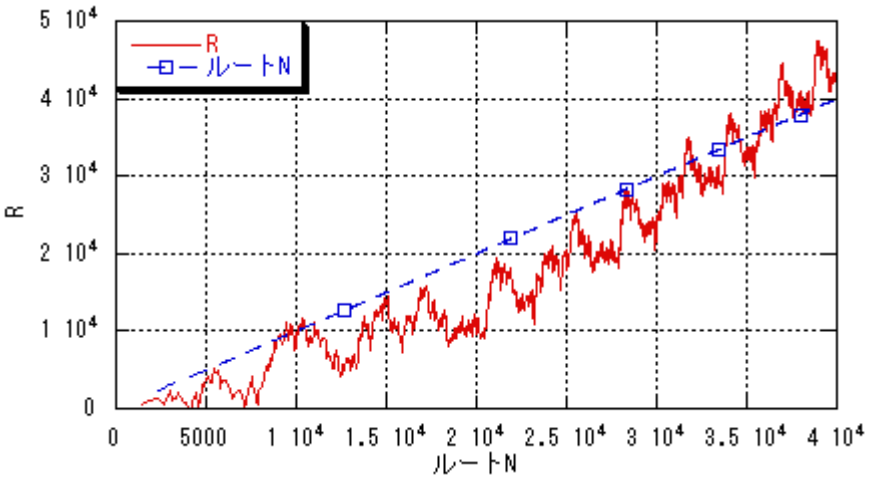


図 3.3　$R \fallingdotseq \sqrt{N}$ のグラフ　N＝$16\times10^8$

図 3.3　　を見ると、試行回数を増やすと理想的な直線に近づいているのが分かる。

つまり、今回定めた数値

M=2147483647

a=1229

b=9378

$x_0$ =57405839

は、良い値であると考えた。

## ４．　結果

混合合同法と乗算合同法において、一様性においては同程度の精度を示したが、周期においては混合合同法の方が長いことがわかった。よって、シミュレーションには混合合同法を使用するのが良いと考える。法Mが小さい場合には、周期が短くなるだけでなく、一様性においても問題がある。また、今回の研究では、合同法の周期は、$2^n$で表されることがわかった。

今回の研究の結果、混合合同法に用いる値として、

係数　M=2147483674$\left(=2^{31}-1\right)$

乗数　a=1229

加数　b=9378

初期値　$x_0=57405839$

が適当であると、1 次元ランダム・ウォークのシミュレーション結果から判断。さらに、その値の妥当性は 2 次元ランダム・ウォークの結果からも裏付けられた。

## 5．まとめ

【乱数】

乱数とは、サイコロを振って得られる数値のように、得られるどの数値の間にも何の関係も無い数値のことをいう。また、どの数値も等確立で出現する乱数を一様乱数という。しかし、実際にシミュレーションなどでよく使用される乱数は算術乱数であり、これは、計算により乱数が得られるので、どの数値の間にも何の関係も無いとはいえない。よって、このような乱数を擬似乱数という。

擬似乱数の満たすべき条件として以下のことが上げられる。

(a)　乱数発生法は多数個の乱数を速やかに発生できるものであること。

(b)　発生する乱数に周期があるなら、その周期は十分に長いものでなくてはならない。

(c)　再現性がある。初期条件が分かっていれば、いつ何時乱数を発生させても同じ結果が得られる。

(e)　発生した乱数は、良好な統計的性質を持たなくてはならない。

合同法乱数

一様乱数の生成方法として最も広く使われてきたのは、レーマー（Lehmer）が 1948 年頃に発表した線形合同法（linear congruential method）であろう。これは、漸化式

$$x_n = a \cdot x_{n-1} + b \quad (\text{mod M})$$

で得られる非負整数列 $x_n$ を、Mで割ることにより区間 $[0,1)$ の乱数を、$M/10$ で割ることにより区間 $[0,9]$ の乱数を得る。$b = 0$ の場合を乗算合同法、$b \neq 0$ 混合合同法と呼ぶ。a は乗数、ｂは加数、M は係数を示し、n=1,2,3,･･･である。

また、n=1 のとき右辺の $x_{n-1}$ は $x_0$ であり、この $x_0$ は初期値である。

乗算合同法

乗算合同法とは、２数の積をある数で割った余りをとる方法である。漸化式

$$x_n = a \cdot x_{n-1} \quad (\text{mod M})$$

で表される。この漸化式により、数列 $x_1, x_2, x_3$･･･を発生させる。

混合合同法

混合合同法とは、2 数の積にある数を加えたものをある数で割った余りをとる方法である。漸化式

$$x_n = a \cdot x_{n-1} + b \quad (\text{mod M})$$

で表される。この漸化式により、数列 $x_1, x_2, x_3 \cdots$ を発生させる。

区間 $[0,9]$ の乱数 $y_n$ は、$x_n$ を M を 10 で割った値で割り、得られた値の整数部をもちいることによって得られる。

区間 $[0,1)$ の乱数 $z_n$ は、$x_n$ を M で割ることにより得られる。

【乱数の検定】

乱数には、いくつかの検定法があるが、唯一つの検定法でその擬似乱数が理想的な乱数に近いかどうかを判断するのは危険である。擬似乱数においては、一様性や周期などにおいて十分検定を行う必要がある。例えば、一様性の検定として

平均　$\sum_{n-1}^{N} \frac{x_n}{N} = \frac{1}{2}$

2 次積率　$\sum_{n-1}^{N} \frac{x_n^2}{N} = \frac{1}{3}$

3 次積率　$\sum_{n-1}^{N} \frac{x_n^3}{N} = \frac{1}{4}$

が上げられるが、この検定の結果、良好な値が得られても周期が極端に短い場合も考えうるのである。実際に、２．７　合同法の比較　において図 2.7.1.3 において一様性が良好な値に収束しているが、周期は図 2.7.2.3 に示されている通り大変短い（実際には、128）ということも起こりうるのである。

【ランダム・ウォーク】

ランダム･ウォークのシミュレーションは、酔っ払いが千鳥足で歩いているようなモデルを考える。つまり、一歩ごとに方向が定まらず、どの歩も独立し

ている。このようなシミュレーションは、ブラウン運動（Brownian motion）や、金属中の電子の輸送現象のような、粒子のランダムな動きなど物理現象のシミュレーションに用いられる。

また、2 次元のランダム・ウォークの評価法として、

$$R \fallingdotseq \sqrt{N}$$

（Rは原点からの距離、Nは試行回数）

がある。

本稿では、1 次元ランダム・ウォークを乱数の新しい検定法として採用し、それから得られたパラメータを用いることによって、理論的に推定される 2 次元ランダム･ウォークの距離の 2 乗と試行回数の関係式の再現を可能にさせることに成功した。

## ６．　参考文献

今回、この論文を書くにあたり、参考とした文献を紹介します。

【１】 定道宏監修、布上康夫・中原明宏共著、
文系のための C 基礎実習－算法とプログラミング－（オーム社、1994）

【２】 宮武修・脇本和昌共著、　乱数とモンテカルロ法（森北出版、1978）

【３】 伏見正則著、　乱数（東京大学出版会、1994 第二版）

【４】 津田孝夫、　モンテカルロ法とシミュレーション（培風館、1995 三訂版）

【５】 小柳義夫監訳、狩野覚・春日隆・善甫康成訳、
計算物理学　基礎編（朝倉書店、2001 初版）

## 7．　謝辞

本論文を書くにあたり、指導教員である山本哲也教授に多くのご助言、ご指導を頂いた。また、院生の浜小路欣大さんには資料のまとめ方などのご指導を頂いた。山本哲也教授と浜小路欣大さん、また参考にさせていただいた文献の著者・訳者の方々に心から感謝の意を表したい。

そして、私の大学進学を支援し応援してくれた父・春樹、母・明美、私の体調を私以上に気にかけてくれる祖母・ひで子に、この論文の完成を知らせると共に、心からの感謝の意を表したい。

付録

---

プログラム１．　　乗算合同法

---

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main(void)
        {
         double x,rand,rand01,M;
         double base(double);
         int m,rand09;

         x=257;                                          /*X0*/
        M=34768;                                         /*mod M*/

        rand09=0;
        rand01=0;
        rand=0;

        for(m=1;m<10001;m++) {

                                x=base(x);

                                rand=(x/M)*10;
                                rand01=x/M;              /*[0,1)*/

                                rand09=rand;             /*[0,9]*/

                                printf("%d¥t%d¥t%f¥n",m,rand09,rand01);
                                    }
         return(0);
       }
```

```
double base(double x)                                   /*Xn*/
    {
      double a=3045,M=34768,x1,x2;
      x1=0;
      x2=0;
      x1=(a*x)/M;
      x1=(int)x1;
      x2=(double)x1*M;
      x=(a*x)-x2;

      return x;
}
```

---

プログラム２．　　混合合同法

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main(void)
        {
         double x,rand,rand01,M;
         double base(double);
         int m,rand09;

         x=257;                                          /*X0*/
        M=34768;                                         /*mod M*/

        rand09=0;
        rand01=0;
        rand=0;

        for(m=1;m<10001;m++) {

                                x=base(x);

                                rand=(x/M)*10;
                                rand01=x/M;              /*[0,1)*/

                                rand09=rand;             /*[0,9]*/

                                printf("%d¥t%d¥t%f¥n",m,rand09,rand01);
                                  }

        return(0);
```

```
        }

double base(double x)                                   /*Xn*/
    {
      double a=3045,b=30120,M=34768,x1,x2;
      x1=0;
      x2=0;
      x1=(a*x+b)/M;
      x1=(int)x1;
      x2=(double)x1*M;
      x=(a*x+b)-x2;

      return x;
}
```

---

プログラム 3　　乱数の発生と検定　　混合合同法

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main(void)
        {
         double x,rand,rand01,M;
         double base(double);
         double   kei,heikin,nizi,nizisekiritu,sanzi,sanzisekiritu;
         int m,rand09;

         x=257;                                                    /*X0*/
        M=34768;                                                   /*mod M*/
        rand09=0;
        rand01=0;
        rand=0;
        kei=0;
        heikin=0;
        nizi=0;
        nizisekiritu=0;
        sanzi=0;
        sanzisekiritu=0;

        for(m=1;m<10001;m++) {
                              x=base(x);
                              rand=(x/M)*10;
                              rand01=x/M;
                              rand09=rand;

                    /*        ritu        */
                              kei=kei+rand01;
```

```
                                heikin=kei/(double)m;
                                nizi=nizi+rand01*rand01;
                                nizisekiritu=nizi/(double)m;
                                sanzi=sanzi+rand01*rand01*rand01;
                                sanzisekiritu=sanzi/(double)m;

            printf("%d¥t%d¥t%f¥t%f¥t%f¥n",m,rand09,rand01,
                                                heikin,nizisekiritu,sanzisekiritu);
                                        }
          return(0);
         }

double base(double x)                                   /*Xn*/
    {
      double a=3045,b=9378,M=34768,x1,x2;
      x1=0;
      x2=0;
      x1=(a*x+b)/M;
      x1=(int)x1;
      x2=(double)x1*M;
      x=(a*x+b)-x2;

      return x;
}
```

---

プログラム 4　　乱数の発生と検定　　乗算合同法

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main(void)
        {
         double x,rand,rand01,M;
         double base(double);
         double   kei,heikin,nizi,nizisekiritu,sanzi,sanzisekiritu;
         int m,rand09;

         x=257;                                                    /*X0*/
        M=34768;                                                   /*mod M*/
        rand09=0;
        rand01=0;
        rand=0;
        kei=0;
        heikin=0;
        nizi=0;
        nizisekiritu=0;
        sanzi=0;
        sanzisekiritu=0;

        for(m=1;m<10001;m++) {
                              x=base(x);
                              rand=(x/M)*10;
                              rand01=x/M;
                              rand09=rand;

                    /*        ritu        */
                              kei=kei+rand01;
                              heikin=kei/(double)m;
                              nizi=nizi+rand01*rand01;
```

```
                    nizisekiritu=nizi/(double)m;
                    sanzi=sanzi+rand01*rand01*rand01;
                    sanzisekiritu=sanzi/(double)m;

        printf("%d¥t%d¥t%f¥t%f¥t%f¥n",m,rand09,rand01,
                                    heikin,nizisekiritu,sanzisekiritu);
                }
      return(0);
     }

double base(double x)                                   /*Xn*/
   {
    double a=3045,M=34768,x1,x2;
    x1=0;
    x2=0;
    x1=(a*x)/M;
    x1=(int)x1;
    x2=(double)x1*M;
    x=(a*x)-x2;

    return x;
}
```

---

プログラム５　シミュレーション　１次元ランダムウォーク

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main(void)
        {
         double x,rand,M;
         double base(double);
         double iti ;
         int m,rand09,amari;

         x=257;                                                    /*X0*/
        M=34768;                                                   /*mod M*/
        rand09=0;
        rand01=0;
        rand=0;

        for(m=1;m<10001;m++) {
              x=base(x);
              rand=(x/M)*10;
              rand09=rand;
              amari=rand09%2;

              if(amari==1){
                                    iti=iti+1;
                                     }

              else{
                                    iti=iti-1;
                                       }


     printf("%d¥t%d¥n",m,iti);
```

```
            }
          return(0);
        }

double base(double x)                                   /*Xn*/
    {
      double a=3045,b=9378,M=34768,x1,x2;
      x1=0;
      x2=0;
      x1=(a*x+b)/M;
      x1=(int)x1;
      x2=(double)x1*M;
      x=(a*x+b)-x2;

      return x;
}
```

---

プログラム６　シミュレーション　２次元ランダムウォーク

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

main(void)
        {
         double x,rand01,M;
         double base(double);
         double itix,itiy,R,N ;
         int m,n;

         x=257;                                                    /*X0*/
        M=34768;                                                   /*mod M*/
        rand01=0;

        for(m=1;m<1600000001;m++) {
              x=base(x);
              rand01=x/M;

              if(rand01>=0 && rand01<0.249999){
                                    itiy=itiy+1;
                                    itix=itix;
                                     }
              else if(rand01>=0.249999 && rand01<0.499999){
                                    itiy=itiy-1;
                                    itix=itix;
                                      }
              else if(rand01>=0.499999 && rand01<0.749998){
                                    itiy=itiy;
                                    itix=itix+1;
```

```
                                        }
            else{
                                    itiy=itiy;
                                    itix=itix-1;
                                        }

      R=sqrt((itix*itix)+(itiy*itiy));
      N=sqrt((double)m);

      if(m==n*1000000){
      n=n+1;
      printf("%d¥t%f¥t%f¥n",m,R,N);
                                    }
            }
        return(0);
       }

double base(double x)                                   /*Xn*/
    {
      double a=3045,b=9378,M=34768,x1,x2;
      x1=0;
      x2=0;
      x1=(a*x+b)/M;
      x1=(int)x1;
      x2=(double)x1*M;
      x=(a*x+b)-x2;

      return x;
}
```